UNCLASSIFIED

Defense Technical Information Center Compilation Part Notice

ADP023867

TITLE: Optimizing Finite Element Programs on the Cray X1 Using Coloring Schemes

DISTRIBUTION: Approved for public release, distribution unlimited

This paper is part of the following report:

TITLE: Proceedings of the HPCMP Users Group Conference 2004. DoD High Performance Computing Modernization Program [HPCMP] held in Williamsburg, Virginia on 7-11 June 2004

To order the complete compilation report, use: ADA492363

The component part is provided here to allow users access to individually authored sections of proceedings, annals, symposia, etc. However, the component should be considered within the context of the overall compilation report and not as a stand-alone technical report.

The following component part numbers comprise the compilation report:
ADP023820 thru ADP023869

UNCLASSIFIED

Optimizing Finite Element Programs on the Cray X1 Using Coloring Schemes

Fred T. Tracy
USACE Engineering Research and Development Center (ERDC), Vicksburg, MS
Fred.T.Tracy@erdc.usace.army.mil

Abstract

Using the Environmental Quality Modeling program FEMWATER as a test-bed code, 27 percent of the time needed to run a given groundwater flow application on the ERDC Cray X1 using four multistream processors (MSPs) was spent assembling the global stiffness matrix. This poor performance is because the above code cannot multistream without help. The technique of "coloring" the elements makes it possible to multistream this section of the code, thus taking advantage of the hardware capability of the machine. Coloring for assembling the global stiffness matrix involves dividing the elements into different groups such that no node point touches any elements with the same color. This paper will present a simple coloring algorithm in FORTRAN and show how it was implemented into FEMWATER to achieve multistreaming on the ERDC Cray X1. It will then give a detailed description on how the program was modified, what compiler options were used, and what compiler directives worked best. Finally, timing results will be given. Some programs that have good MPI (or equivalent) communication are better suited for running in the single-streamed processor (SSP) mode. In the SSP mode, coloring of the elements is not needed for assembling the global stiffness matrix. Timings for running in the SSP mode will be shown, too.

1. Introduction

Vectorizing and multistreaming application codes on the Cray X1 is essential to good performance. Because of this, bottlenecks in code performance can occur in surprising places. Sometimes the algorithm is just not suited for the X1. Many times, however, special techniques and algorithms can be applied to remedy the situation. This paper illustrates one such example of where "coloring" the finite elements into separate groups can significantly help. Assembling the element stiffness matrices into the global stiffness matrix originally took 27 percent of the total run time. This part of the program now runs eight times faster when using the coloring scheme described in this paper.

2. FEMWATER

FEMWATER[4,5] is a standard Galerkin finite element program for flow and transport. METIS[2] was used to partition the mesh. A conjugate gradient, iterative solver with an incomplete lower-upper preconditioner[1] was used to solve the resulting system of linear equations obtained from a Picard iteration of the nonlinear equations. The ghost node updates are done in this study using MPI.

2.1. Test Problem.

Figure 1 illustrates the top view of a typical three-dimensional (3-D) finite element mesh for a remediation study. Several layers are used to model the soil layers underneath this surface. The mesh has 102,996 nodes and 187,902 3-D prism elements. Runs using two and four times the original number of elements are also done.

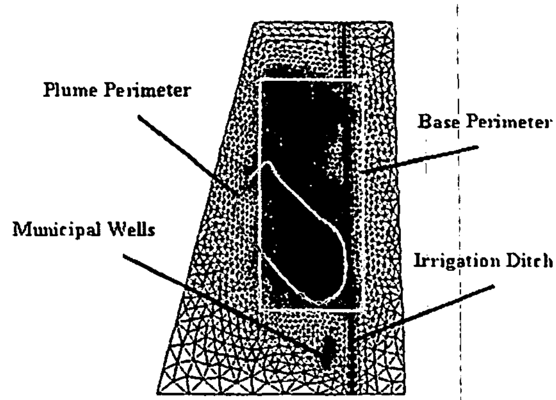


Figure 1. FEMWATER test problem

2.2. Assembly of Global Stiffness Matrix.

Figure 2 shows the original assembly process. Subroutine `fq468` computes element stiffness matrix information. The `i` and `j` loops are from 1 to 6, representing prism elements. `ni` is the node or row number in the global stiffness matrix, and `jj` is the column number in the global stiffness matrix. These indices were reversed in storage to have stride 1 computations. Finally, a search for `jj` must be done inside the `j` loop. Figure 3 shows how coloring modifies the basic routine. This modified version is described in more detail below.

3. Coloring

Coloring for assembling the global stiffness matrix involves dividing the elements into different groups such that no node point touches any elements with the same color. Figure 4 illustrates the process of generating the different groups through coloring. The algorithm presented in this paper written in FORTRAN (see Figure 5) is very similar to that discussed in Reference 3. Starting with color 1 (blue), an element is first painted. Then all elements that touch this element are marked for not allowing blue. A second blue element is done, again marking elements touching the new blue element as not allowing blue. After all the possible blue elements have been painted, this procedure is repeated for the second color (red). This algorithm is completed when all elements have been provided a color. Finally, all elements that contain the same color belong to the same group.

The code in Figure 2 can now be changed to the code in Figure 3. The do loop over mm can now be multistreamed and vectorized. This will now be described.

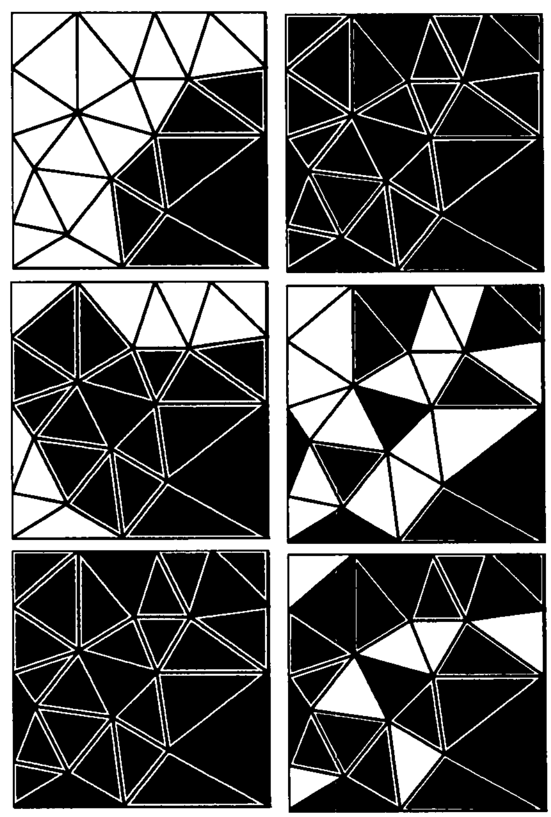
```
 do m = 1, nel
   ..........
   call fq468 (m)
   ..........
   do i = 1, 6
     ni = iem(i)
     do j = 1, 6
       ..........
c Search here to find the column number jj
for the row ni.
       ..........
       global_stiff(jj, ni) =
global_stiff(jj, ni) +
&          element_stiff(i, j)
     end do
   end do
 end do
```

Figure 2. Original assembly process before coloring

```
do ig = 1, no_of_groups
   mm1 = starting_element_of_group(ig)
   mm2 = starting_element_of_group(ig + 1)
- 1
   do mm = mm1, mm2
     m = mth_element_of_group_ig(mm)
     ..........
     call fq468 (m)
     ..........
     do i = 1, 6
       ni = iem(i)
       do j = 1, 6
       ..........
c Search here to find the column number jj
for the row ni.
       ..........
         global_stiff(jj, ni) =
global_stiff(jj, ni) +
&          element_stiff(i, j)
       end do
     end do
   end do
 end do
```

Figure 3. Modified assembly process after coloring

Figure 4. Coloring the elements



```
C Initialize colors of all elements to
zero.
 icolor(1 : no_of_eements) = 0
 i = 0
 iquit = 0

 do while ((i .lt. max_no_colors) .and.
(iquit .eq. 0))

   i = i + 1

c Go through all node points to try to
color elements with i.
   do n = 1, no_node_points
     mfound = 0
     do m = 1,
no_elements_connected_to_node_n
       if {an element has color i} mfound =
1
     end do
     if (mfound .eq. 0) then
       iel_first_zero = 0
       do m = 1,
no_elements_connected_to_node_n
         {If element m has icolor = 0, set
iel_first_zero = m}
       end do
       if (iel_first_zero .ne. 0) then
c Set element to color i here.
         icolor(iel_first_zero) = i
c Set the color of all elements touching
iel_first_zero to -1
c if no color has been assigned yet.
         do j = 1,
no_nodes_of_iel_first_zero
           do k = 1,
no_elements_connected_to_node_j
             iel_k =
kth_element_connected_to_node_n
             if (icolor(iel_k) .eq. 0)
icolor(iel_k) = - 1
           end do
         end do
       end if
     end if
   end do

c Clean up -1's for the next color and
check for termination.
   iquit = 1
   do n = 1, no_of_elements
     if (icolor(n) .eq. -1) then
       icolor(n) = 0
       iquit = 0
     end if
   end do

 end do
```

Figure 5. Coloring algorithm in FORTRAN

4. Multistreaming and Vectorizing

Space allows a brief description of what was then done to achieve vectorization and multistreaming of the code. Figure 6 shows a portion of the new algorithm with compiler notation provided. This listing was generated using the command,

```
103. 1---------< do ig = 1, nog

105. 1             mm1 = istg(ig)
106. 1             mm2 = istg(ig + 1) - 1

108. 1            !csd$ parallel do private
(mm, m, node, mtyp,
109. 1            !csd$& alp, por, iq, iem,
ni, jq, nj)

115. 1 M-------<   do mm = mm1, mm2

117. 1 M               m = ielg(mm)
118. 1 M               NODE = IJNOD(M)
120. 1 M               MTYP = IE(M, 9)
121. 1 M               ALP = PROPF(7, MTYP)
122. 1 M               POR = PROPF(8, MTYP)
124. 1 M MVs---<       DO IQ = 1, NODE
125. 1 M MVs             IEM(IQ) = IE(M, IQ)
126. 1 M MVs--->       END DO
         Intermediate computations.
276. 1 M          !dir$ concurrent
277. 1 M MV----<       DO IQ = 1, NODE
278. 1 M MV              NI = IEM(IQ)
279. 1 M MV               RLD(NI) = RLD(NI) +
RQ(IQ)
280. 1 M MV Mr-<         DO JQ = 1, NODE
281. 1 M MV Mr             NJ = IEM(JQ)
282. 1 M MV Mr                  QA(IQ, JQ) =
QA(IQ, JQ) * DELTI
285. 1 M MV Mr                RLD(NI) = RLD(NI)
+ (QA(IQ, JQ)-
286. 1 M MV Mr   &             W2 * QB(IQ, JQ))
* HP(NJ)
298. 1 M MV Mr                cmatrx(iw(iq, jq,
m), ni) =
299. 1 M MV Mr    &              cmatrx(iw(iq,
jq, m), ni) +
300. 1 M MV Mr   &            qa(iq, jq) + w1
* qb(iq, jq)
301. 1 M MV Mr->           END DO
302. 1 M MV---->         END DO

305. 1 M------->    end do
306. 1           !csd$ end parallel do

310. 1---------> end do
```

Figure 6. Assembly process showing compiler notations

```
ftn -c -O3,aggress -rm fasemb_x1.f
```

The meaning of the notations are as follows:

```
M-Multistreamed      r-unrolled
V-Vectorized         s-short loop
```

The changes to the original code are as follows:

1. Change loops to use the coloring algorithm (Figure 3).
2. Add compiler directives (lines 108–109, 276, 306 of Figure 6). Sometimes the Cray streaming directives (csd) work, and sometimes the `!dir concurrent` works. In this case, using both of them was required.
3. The search to find the column number `jj` for the row `ni` is a bottleneck. Therefore, a one-time calculation was made to avoid this search (variable `iw` in lines 298–300 of Figure 6).
4. Manually inline subroutine `fq468`.
5. Remove error print statements until after the important loops.

An alternate way of dealing with these loops is to get them to unroll as discussed in Reference 3. To test this, the loops were manually unrolled creating hundreds of lines as illustrated in Figure 7.

```
108. 1      !csd$ parallel do private (mm,
m, node, mtyp,
109. 1      !csd$& alp, por, iq, iem, ni,
jq, nj)

115. 1 MV-< do mm = mm1, mm2

117. 1 MV     m = ielg(mm)
------------------
127. 1 MV     IEM(1) = IE(M, 1)
128. 1 MV     IEM(2) = IE(M, 2)
129. 1 MV     IEM(3) = IE(M, 3)
130. 1 MV     IEM(4) = IE(M, 4)
------------------

305. 1 MV-> end do

306. 1      !csd$ end parallel do
```

Figure 7. Assembly process using unrolling

The timings were the same as that achieved from the way Figure 6 shows it; therefore, the results in Figure 6 represent the best effort for these loops.

5. Performance Results

Table 1 shows timing results for the original mesh, twice the size of the original mesh, eight times the size of the original mesh, and sixteen times the size of the original mesh. For each problem, the modified MSP version of the assembly process was sped up approximately eight times. The problems were also run using 16 SSPs. Despite the good MPI communication, the modified MSP version runs four times faster than the original SSP version. Coloring of elements is not needed in the SSP mode, because this is for multistreaming only.

Table 1. X1 timings (sec) for 4 MSPs or 16 SSPs

Nodes	102,996	197,409	763,887	1,519,191
Elements	187,902	375,804	1,503,216	3,006,432
Mesh size increase	1X	2X	8X	16X
Total time – MSP – original	745	1,419	5,444	11,171
Total time – SSP – original	257	493	1,958	3,636
Assembly time– MSP – original	187	379	1,458	2,733
Assembly time – MSP – modified	23	46	176	330
Assembly MSP Ratio	8.1	8.2	8.3	8.3
Assembly time – SSP – original	81	158	604	1,165

Acknowledgment

This work was supported in part by a grant of computer time from the DoD High Performance Computing Modernization Program at the ERDC MSRC, Information Technology Laboratory, Vicksburg, MS.

References

1. Dongara, J.J., D.C. Sorensen, and H.A. van der Vorst, "Numerical Linear Algebra for High-Performance Computers." SIAM, Philadelphia, PA, 1998, p. 203.

2. Karypis, G., METIS (computer program library), http://www.users.cs.umn.edu/~karypis/metis/metis.html, University of Minnesota, MN, 2004.

3. Kornkven, E. and A. Johnson., SC2003 Tutorial M-8, "Vector Performance Programming." Phoenix, AR, 2003.

4. Lin, H.J., D.R. Richards, G.T. Yeh, J. Cheng, H. Cheng, and N.L. Jones, "FEMWATER: A Three-Dimensional Finite Element Computer Model for Simulating Density-Dependent Flow and Transport in Variably Saturated Media." *Technical*

Report CHL-97-12, US Army Engineer Research and Development Center (ERDC), Vicksburg, MS, 1997.

5. Tracy, F.T., C.A. Talbot, J.P. Holland, S.J. Turnbull, T.L. McGehee, and B.P. Donnell, "The Application of the Parallelized Groundwater Model FEMWATER to a Deep Mine Project and the Remediation of a Large Military Site." *DoD HPCMP Users Group Conference Proceedings*, Monterey, CA, 1999.